Όλο το υλικό για την χρήση και προγραμματισμό των Alpha-2 μπορεί να βρεθεί στον εξής σύνδεσμο:

https://drive.google.com/open?id=1StK6oF_1RRPceogaGr-lNstyjxMuR_H5

Για την επικοινωνία υπολογιστή – ρομπότ θα χρησιμοποιηθεί το πρόγραμμα ADB. Για να εγκατασταθεί το ADB σε περιβάλλον Linux χρησιμοποιήστε την εντολή:

```
sudo apt update
sudo apt-get install android-tools-adb android-tools-fastboot
```

Για να λειτουργήσει σωστά ένα .apk πρόγραμμα στο ρομπότ, πρέπει αυτό να είναι συνδεδεμένο στο internet. Προκειμένου να ρυθμίσουμε τα network interfaces του ρομπότ προκειμένου να δώσουμε τα στοιχεία της Wi-Fi σύνδεσής μας, εκτελούμε τα παρακάτω:

```
adb root
adb pull /data/misc/wifi/wpa_supplicant.conf
```

…τροποποιούμε κατάλληλα το αρχείο wpa_supplicant.conf

```
adb push wpa_supplicant.conf /data/misc/wifi
```

Η σύνδεση στο internet απαιτείται προκειμένου να υπάρχει συνεχής επικοινωνία του ρομπότ με τον server της ubtech. Γι’ αυτό το λόγο, προκειμένου ο server να εντοπίζεται στην σωστή τοποθεσία, πρέπει να ελέγξουμε ότι στο αρχείο /etc/hosts περιέχεται η παρακάτω γραμμή:

```
112.74.140.14   dev.ubtrobot.com
```

Όπου 112.74.140.14 το ip που προκύπτει εκτελώντας την εντολή:

```
ping developer.ubtrobot.com
```

Σε περίπτωση που η γραμμή αυτή δεν περιέχεται, τότε εκτελούμε τις εξής εντολές προκειμένου να την προσθέσουμε:

```
adb root
adb remount
adb pull /etc/hosts
```

…προσθέτουμε την επιθυμητή γραμμή στο αρχείο hosts

```
adb push hosts /etc/wifi
adb shell
cd /etc/wifi
cp hosts ../
rm hosts
cd ..
cat hosts
```

(επαλήθευση ότι είναι σωστό)

Τέλος, προκειμένου να αναπτυχθεί και να τρέξει η δικιά μας εφαρμογή, πρέπει να μην τρέχει ταυτόχρονα κάποια άλλη που να χρησιμοποιεί τους πόρους του ρομπότ. Υπάρχει προεγκατεστημένο το alphaenglishchat, το οποίο και πρέπει να απεγκατασταθεί:

```
adb uninstall com.ubtechinc.alphaenglishchat
```

Σε περίπτωση που θέλουμε να το ξαναεγκαταστήσουμε, τρέχουμε την εντολή:
`adb install alphaenglishchat.apk`
με το αρχείο alphaenglishchat.apk να βρίσκεται στο τρέχον directory.

Στη συνέχεια, το εκτελούμε με την εντολή:
`adb shell am start -n com.ubtechinc.alphaenglishchat/.activity.ChatActivity`
(η συγκεκριμένη εφαρμογή εκτελείται και αυτόματα κατά την εκκίνηση, μετά που θα γίνει η 1η επανεκκίνηση του ρομπότ).

Όμοια χειριζόμαστε και οποιοδήποτε άλλο αρχείο .apk (όπως την εφαρμογή που θα αναπτύξουμε) για εγκατάσταση/απεγκατάσταση/εκτέλεση:

```
adb install alpha2demo.apk
adb uninstall com.ubtech.alpha2demo
adb shell am start -n com.ubtech.alpha2demo/.KouretesActivity
adb shell am force-stop com.ubtech.alpha2demo
```

Συνοψίζοντας, για να προετοιμαστεί το ρομπότ να δεχτεί την εφαρμογή μας πρέπει:

- Να είναι συνδεδεμένο στο internet.
- Να περιλαμβάνει την κατάλληλη καταχώρηση στο /etc/hosts
- Να μη βρίσκεται εγκατεστημένη κάποια εφαρμογή όπως το alphaenglishchat που να χρησιμοποιεί πόρους.

Στη συνέχεια, φορτώνουμε το project ‘alpha2demo’(alpha2demo_kouretes.zip) στο Android Studio. Για να εκτελεστεί η εφαρμογή μας, μπορούμε να την τρέξουμε είτε κατευθείαν μέσα από το Android Studio (οπότε θα μπορούμε να παρακολουθούμε και όλα τα διαγνωστικά μηνύματα στο παράθυρο Logcat) είτε να παράξουμε ένα αρχείο .apk και να το τρέξουμε σύμφωνα με τις εντολές παραπάνω. Παρακάτω θα υποθέσουμε ότι ακολουθείται ο 1ος τρόπος.

Το Main Activity της εφαρμογής (το οποίο θα εκτελείται κατά την εκκίνησή της και ορίζεται στο *AndroidManifest.xml*) είναι το *KouretesActivity.java*. Όλη η ανάπτυξη κώδικα θα γίνεται μέσα σε αυτό το activity, και τα υπόλοιπα έχουν παραμείνει στο project μόνο ως σημείο αναφοράς/βοήθειας για τμήματα κώδικα (αλλά δε χρησιμοποιούνται στην πράξη). Αυτή η αναφορά έχει ως στόχο την ανάπτυξη μιας αυτόνομης συμπεριφοράς στο ρομπότ χωρίς επικοινωνία με εφαρμογή σε κάποιο κινητό. Επομένως, η αλληλεπίδραση με το χρήστη θα γίνεται μέσω ομιλίας.

Η ανάπτυξη της εφαρμογής γίνεται με κώδικα σε στυλ Android. Οι κλήσεις από και προς τους πόρους του ρομπότ γίνονται μέσω των συναρτήσεων/μεταβλητών/σταθερών που ορίζονται στο συνοδευτικό αρχείο *ubtechalpha2robot.jar*, και πρέπει να συμπεριλαμβάνεται επίσης στο project.

Προκειμένου να διευκολυνθεί ένας χρήστης και να μη χρειαστεί να εμβαθύνει στις λεπτομέρειες του δοθέντος interface, ο κώδικας έχει χωριστεί σε τμήματα:

Το **1ο μέρος** (με ονομασία “TECHNICAL STUFF”) περιέχει λεπτομέρειες της υλοποίησης ορισμένες αρκετά γενικά και με τέτοιο τρόπο, ώστε για ανάπτυξη απλών εφαρμογών να μη χρειάζονται κάποια τροποποίηση. Επομένως, ένας απλός προγραμματιστής δε χρειάζεται να ασχοληθεί καθόλου με αυτό το τμήμα του κώδικα (που είναι και το μεγαλύτερο).

Το **2ο μέρος** του κώδικα (με ονομασία “ADVANCED STUFF”) περιέχει τις μεθόδους *onSpeechRecognition()* και *calculateConfidence().* Συνήθως, αυτές οι μέθοδοι είναι αρκετά γενικά ορισμένες ώστε να μη χρειάζονται τροποποιήσεις, αλλά ανάλογα με τις απαιτήσεις κάθε project ίσως χρειαστούν κάποιες αλλαγές ορισμένες φορές. Ένας απλός προγραμματιστής δε χρειάζεται να ασχοληθεί ούτε με αυτό το τμήμα, παρά μόνο αν προκύψει στην πορεία η ανάγκη αυτή.

Η μέθοδος *calculateConfidence()* παίρνει ως ορίσματα 2 Strings, το ληφθέν από τη Speech-To-Text μετατροπή και κάποιο ήδη υπάρχον στον κώδικα και τα συγκρίνει ως προς την ομοιότητά τους. Τελικά, επιστρέφει κάποια τιμή τύπου double (ανάμεσα στο 0 και το 2) που υποδεικνύει το ποσοστό της ομοιότητας αυτής. Ο αλγόριθμος υπολογισμού είναι πολύ απλός και επιδέχεται βελτιώσεις, αλλά γενικά είναι αρκετά αποτελεσματικός. Στην τρέχουσα μορφή του, πρόβλημα θα προκύψει στην περίπτωση που υπάρχουν 2 ερωτήσεις που αποτελούνται από τις ίδιες (ή παρόμοιες) λέξεις αλλά με διαφορετική σειρά.

Η μέθοδος *onSpeechRecognition()* παίρνει ως όρισμα 1 String, αυτό το οποίο ελήφθη από τη Speech-To-Text μετατροπή. Στη συνέχεια, χρησιμοποιώντας την *calculateConfidence(),* θα υπολογίσει την πιο πιθανή αντιστοιχία ανάμεσα σε ήδη υπάρχοντες ερωτήσεις στον κώδικα. Τελικά, όταν αποφασίσει ποια είναι η πιο πιθανή ερώτηση, θα καλέσει την *findAppropriateResponse()* η οποία θα αποφασίσει για την απάντηση. Η *findAppropriateResponse()* δε θα κληθεί (άρα το ρομπότ δε θα αντιδράσει) εφόσον η υπολογισθείσα confidence είναι μικρότερη του threshold.

Το **3ο μέρος** του κώδικα (με ονομασία “STANDARD STUFF”) είναι το τμήμα του κώδικα που θα αναπτυχθεί μια συνηθισμένη εφαρμογή. Αποτελείται από τα εξής πεδία:

- Μεταβλητή USE_LOOK_AT_SPEAKER_DIRECTION: Ανάλογα την τιμή της, το ρομπότ θα γυρνάει το κεφάλι (στον οριζόντιο άξονα) προσπαθώντας να κοιτάζει προς την κατεύθυνση της ομιλίας.
- Μεταβλητή USE_ACTIONS: Ανάλογα την τιμή της, θα είναι δυνατή η εκτέλεση actions (=κινήσεων).
- Μεταβλητή USE_SPEECH_RECOGNITION: Ανάλογα την τιμή της, θα είναι δυνατή η ύπαρξη Speech Recognition. Με την τρέχουσα υλοποίηση είναι ο μόνος δυνατός τρόπος αλληλεπίδρασης ενός χρήστη, επομένως η τιμή της θα είναι συνήθως true.
- Μεταβλητή USE_OFFLINE_GRAMMAR: Η τιμή της έχει σημασία μόνο όταν USE_SPEECH_RECOGNITION=true. Εφόσον αυτή η μεταβλητή ισούται με true, η επεξεργασία της φωνής θα γίνεται μέσα στο ρομπότ και θα παράγεται το String με το αποτέλεσμα. Σε αντίθετη περίπτωση, ηχητικό clip με τη φωνή θα αποστέλλεται στον server της ubitech, και το String με το αποτέλεσμα θα αποστέλλεται πίσω ως απάντηση. Προτείνεται η τιμή true, καθώς η ταχύτητα απόκρισης ίσως είναι λίγο πιο γρήγορη, ιδιαίτερα για αργά δίκτυα.

Οι παραπάνω μεταβλητές ενεργοποιούν/απενεργοποιούν βασικά τμήματα της συμπεριφοράς του robot. Συνίσταται κάθε φορά να είναι ενεργοποιημένες μόνο όσες χρησιμοποιούνται, αν και στη πλειοψηφία των περιπτώσεων και οι 4 θα ισούνται με true.

- private ArrayList<Integer> questionHistory: Σε αυτή τη μεταβλητή αποθηκεύεται το ιστορικό των προηγούμενων ερωτήσεων που δέχτηκε το ρομπότ. Για παράδειγμα, αν προηγουμένως δέχτηκε (με χρονολογική σειρά) τις ερωτήσεις στις θέσεις 2,5,3,1 του πίνακα *questions*, τότε το *questionHistory* θα έχει στη θέση 0 τον αριθμό 2, στη θέση 1 τον αριθμό 5 κοκ. Η αποθήκευση/διαχείριση των τιμών γίνεται αυτόματα από την *onSpeechRecognition()* και ο χρήστης μπορεί να τις θεωρεί δεδομένες. Επιπλέον, το μέγιστο μέγεθος αυτού του *ArrayList* μπορεί να ισούται με *maxHistorySize*, και σε περίπτωση επιπλέον στοιχείων διαγράφεται το πιο παλιό (δηλαδή της θέση 0) για να προστεθεί ένα καινούριο (επίσης αυτόματα, από την *OnSpeechRecognition()*).
- int maxHistorySize: Το μέγιστο δυνατό μέγεθος της δομής *questionHistory*.
- String[] questions: Πίνακας που περιέχει όλες τις δυνατές ερωτήσεις που δέχεται το ρομπότ. Αυτή η δομή δεν πρέπει να τροποποιηθεί (να αλλάξει τύπο) εκτός αν αλλάξουν και τα κατάλληλα σημεία στην υλοποίηση της *onSpeechRecognition()*. Φυσικά, ο αριθμός και οι τιμές των στοιχείων μπορούν να αλλάξουν ελεύθερα χωρίς να απαιτείται κάποια παραπάνω αλλαγή.
- String[] answers: Πίνακας που περιέχει όλες τις δυνατές απαντήσεις. Στην τρέχουσα υλοποίηση κάθε στοιχείο (π.χ. της θέσης 0) αντιστοιχεί στην ίδια ερώτηση (της θέσης 0 πάλι). Αυτή η υλοποίηση είναι απλή, αλλά για περίπλοκους διαλόγους δεν αρκεί. Επομένως, ίσως χρειαστεί ανάλογα με τις ανάγκες αλλαγή του τύπου από *String[]* σε πολυδιάστατο πίνακα ή άλλη δομή αποθήκευσης δεδομένων (*Vector*, *ArrayList* κτλ.). Η αλλαγή μπορεί να

γίνει χωρίς προβλήματα, καθώς το μόνο σημείο που χρησιμοποιείται είναι στη μέθοδο *findAppropriateAnswer()* – βλέπε παρακάτω.

- double threshold: Το threshold (τιμές 0-1) για την αναγνώριση μιας ερώτησης.
- void executeActivity(): Ακολουθιακός κώδικας που εκτελείται αμέσως μετά την επιτυχή αρχικοποίηση του ρομπότ. Μπορεί να παραμείνει και άδεια (για δημιουργία ακολουθιακού κώδικα πρέπει να χρησιμοποιηθούν οι onSpeechEnding(), onActionEnding(), βλέπε 3^{ο} παράδειγμα κώδικα παρακάτω).
- void findAppropriateAnswer(): Εκτελείται κάθε φορά που αναγνωρίζεται επιτυχώς μια ερώτηση προς το ρομπότ. Στη μέθοδο αυτή θα γραφτεί η λογική για το πώς δίνονται οι απαντήσεις με βάση την τρέχουσα ερώτηση και το ιστορικό των προηγούμενων ερωτήσεων. Ως ενέργειες του ρομπότ μπορεί να χρησιμοποιηθεί η *startSpeech(String s)* και η *startAction(String s)*. Οι 2 αυτές μέθοδοι είναι non-blocking, δηλαδή θα ολοκληρωθούν (και ο κώδικας θα προχωρήσει παρακάτω) **πριν** η ομιλία/κίνηση του ρομπότ τελειώσει. Επομένως, είναι λάθος να υπάρχουν 2 *startSpeech()* ή 2 *startAction()* κάπου συνεχόμενα. Από την άλλη, δεν είναι λάθος να υπάρχει η *startAction()* αμέσως πριν ή μετά την *startSpeech(),* καθώς είναι απολύτως εφικτό να μιλάει και εκτελεί κάποια κίνηση ταυτόχρονα. Σε περίπτωση που απαιτείται η κλήση της ίδιας συνάρτησης (από τις startSpeech(), startAction() διαδοχικά, πρέπει να χρησιμοποιηθούν οι onSpeechEnding(), onActionEnding() όπως στο 3^{ο} παράδειγμα κώδικα – βλέπε παρακάτω. Η λίστα με τα δυνατά actions τυπώνεται στο Logcat του Android Studio κατά την αρχικοποίηση του ρομπότ, αλλά υπάρχει και στο αρχείο *NAND FLASH/actions/actionInfo.txt* στη μνήμη του ρομπότ.
- void onSpeechEnding(): Εκτελείται αυτόματα με το που ολοκληρωθεί η ομιλία του ρομπότ (και από εκείνη τη στιγμή θα μπορεί να ξεκινήσει μια καινούρια ομιλία).
- void onActionEnding(): Εκτελείται αυτόματα με το που ολοκληρωθεί μια κίνηση του ρομπότ (και από εκείνη τη στιγμή θα μπορεί να ξεκινήσει μια άλλη).

Παραδείγματα κώδικα:

1) Απλός διάλογος

```
//=====================STANDARD STUFF=============================================================
private boolean USE_LOOK_AT_SPEAKER_DIRECTION = true;
private boolean USE_ACTIONS = true;
private boolean USE_SPEECH_RECOGNITION = true;
private boolean USE_OFFLINE_GRAMMAR = true;

private ArrayList<Integer> questionHistory = new ArrayList<Integer>();
private int maxHistorySize = 10;

private String[] questions={"How are you doing", "What is up"};
private String[] answers={"Good. You?", "Nothing. I am bored."};
private double threshold=0.25;  //values between 0 and 1

//This method is executed immediately after the successful initialization of the robot
private void executeActivity(){
    startSpeech("Hello everybody. Ask me something.");
}

private void findAppropriateAnswer(int questionIndex){
    //Write your code here, to create a good dialog
    startSpeech(answers[questionIndex]);
    if(questionIndex==0)
        startAction("ACT0");
    else
        startAction("ACT1");
}

private void onSpeechEnding(){
    Log.v("logging", "onSpeechEnding");
}

private void onActionEnding(String actionName){
    Log.v("logging", "onActionEnding");
}
//================================================================================================
```

2) Περίπλοκος διάλογος

```
//======================STANDARD STUFF============================================================
private boolean USE_LOOK_AT_SPEAKER_DIRECTION = true;
private boolean USE_ACTIONS = true;
private boolean USE_SPEECH_RECOGNITION = true;
private boolean USE_OFFLINE_GRAMMAR = true;

private ArrayList<Integer> questionHistory = new ArrayList<Integer>();
private int maxHistorySize = 10;

private String[] questions={"how are you doing", "how old are you", "why"};
private String[] answers={"not good", "because you are here", "i am not sure", "because nobody told me"};
private double threshold=0.25;  //values between 0 and 1

//This method is executed immediately after the successful initialization of the robot
private void executeActivity(){
    startSpeech("Hello everybody. Ask me something.");
}

private void findAppropriateAnswer(int questionIndex){
    //Write your code here, to create a good dialog
    if(questionIndex==0){
        startSpeech(answers[0]);
        startAction("ACT0");
    }
    else if(questionIndex==1){
        startSpeech(answers[2]);
        startAction("ACT1");
    }
    else if(questionIndex==2 && questionHistory.size()>1 && questionHistory.get(questionHistory.size()-2)==0){
        startSpeech(answers[1]);
        startAction("ACT2");
    }
    else if(questionIndex==2 && questionHistory.size()>1 && questionHistory.get(questionHistory.size()-2)==1){
        startSpeech(answers[3]);
        startAction("ACT3");
    }
}

private void onSpeechEnding(){
    Log.v("logging", "onSpeechEnding");
}

private void onActionEnding(String actionName){
    Log.v("logging", "onActionEnding");
}
//================================================================================================
```

3) Ακολουθιακές ενέργειες

```
//======================STANDARD STUFF=================================================
private boolean USE_LOOK_AT_SPEAKER_DIRECTION = true;
private boolean USE_ACTIONS = true;
private boolean USE_SPEECH_RECOGNITION = true;
private boolean USE_OFFLINE_GRAMMAR = true;

private ArrayList<Integer> questionHistory = new ArrayList<Integer>();
private int maxHistorySize = 10;

private String[] questions={"how are you doing", "how old are you", "why"};
private String[] answers={"not good", "because you are here", "i am not sure", "because nobody told me"};
private double threshold=0.25;  //values between 0 and 1

boolean talk1=false, talk2=false, act3=false, talk4=false, act4=false;
//This method is executed immediately after the successful initialization of the robot
private void executeActivity(){
    startSpeech("Hello everybody.");
    talk1=true;
}

private void findAppropriateAnswer(int questionIndex){
    Log.v("logging", "onFindAppropriateAnswer");
    //Write your code here, to create a good dialog
}

private void onSpeechEnding(){
    Log.v("logging", "onSpeechEnding");
    if(talk1==true){
        talk1=false;
        startSpeech("Now I will talk for a while");
        talk2=true;
    }
    else if(talk2==true){
        talk2=false;
        startAction("ACT0");
        act3=true;
    }
    else if(talk4==true){
        talk4=false;
        if(act4=false) //action completed before speech
            startAction("ACT2");
    }

}

private void onActionEnding(String actionName){
    Log.v("logging", "onActionEnding");

    if(act3==true){
        act3=false;
        startAction("ACT1");
        startSpeech("Now I will talk and act the same time");
        talk4=true;
        act4=true;
    }
    else if(act4==true){
        act4=false;
        if(talk4=false) //speech completed before action
            startSpeech("Now I will talk for last time");
    }
}
//================================================================================================
```